Всероссийская олимпиада школьников по информатике, профиль «Программирование» Вологодская область, 2025-26 учебный год II (муниципальный) этап, 9 - 11 классы

Методические рекомендации по разбору задач

Задача 1. Красивое время

В данной задаче нужно всего лишь аккуратно написать перебор времён в заданном диапазоне (учитывая возможный переход через полночь) и проверку выполнения критериев «красивости». Пример решения на языке Python:

```
def beautiful(h, m):
    if h == m:
        return True
    if h//10 == m%10 and h%10 == m//10:
        return True
    if h//10+1 == h%10 and h%10+1 == m//10 and m//10+1 == m%10:
        return True
    return False

h1 = int(input())
m1 = int(input())
h2 = int(input())
m2 = int(input())
ans = 0
while True:
    if beautiful(h1, m1):
        ans += 1
    if h1 == h2 and m1 == m2:
        break
    m1 += 1
    if m1 == 60:
        m1 = 0
        h1 += 1
        if h1 == 24:
            h1 = 0
print(ans)
```

Задача 2. Сумма делителей

При n = 1 можно вывести любое число и его делитель – оно же само.

При n = 2 решения нет.

Случай n = 3 дан в условии – ответом будет число 6 с делителями 3, 2 и 1.

Дальше осталось догадаться, что в качестве ответа для числа n+1 можно взять удвоенный ответ для числа n. Действительно, пусть m – это ответ для числа n. Тогда возьмём 2*m в качестве ответа для n+1, а к списку делителей добавим m. Пример:

n=4: 12=6+3+2+1

n=5: 24=12+6+3+2+1

n=6: 48=24+12+6+3+2+1, и так далее.

Пример решения на языке Python:

```
n = int(input())
if n == 1:
   print(1)
   print(1)
elif n == 2:
   print(-1)
else:
   x = 3 * 2**(n-2)
   print(x)
   for i in range(n):
      x = (x + 1) // 2
      print(x, end=' ')
```

Задача 3. Реклама

В первой подзадаче можно создать массив, в котором отмечать для каждой секунды, была ли в эту секунду реклама. Затем идти по этому массиву окном длины 3600, вложенным циклом идти по окну и считать, сколько секунд рекламы попало в это окно.

Во второй подзадаче достаточно заметить, что нет необходимости вложенного цикла по окну – ведь при смещении на один шаг один элемент выходит из окна, и один заходит в окно. Пример такого решения на языке Python:

```
a = [0] * 1000000
n = int(input())
T = int(input())
for _ in range(n):
   t1, t2 = map(int, input().split())
   for i in range(t1, t2):
      a[i] = 1
adv = 0
for i in range(3600):
   adv += a[i]
```

```
if adv > T:
   print(0)
else:
   for i in range(1, 1000000 - 3600 + 1):
      adv = adv - a[i-1] + a[i+3599]
      if adv > T:
         print(i)
         break
   else:
       print(-1)
```

Заметим, что здесь else относится к for, а не к if. Оно срабатывает в том случае, если цикл for завершился естественным образом, а не оператором break.

В третьей подзадаче используем метод двух указателей. Для начала решим задачу для случая, если бы нам не обязательно требовалось самое раннее время. Тогда можно считать, что начало искомого окна совпадает с моментом начала какой-то рекламы (так как если начало окна будет внутри рекламы, то можно окно сдвинуть влево на её начало, при этом решение не испортится).

Пусть левый указатель L будет перебирать начальную рекламу в окне. Правый указатель R сдвинем на последнюю рекламу, которая ещё пересекается с нашим окном. Также будем поддерживать суммарную длину всех реклам от L до R-1. Окончательную длину получаем, добавив кусочек от рекламы R, который пересекается с окном (возможно, и всю рекламу R). Как только эта длина стала больше T, ответ найден.

Теперь как из этого решения получить решение с самым ранним временем? Во-первых, если реклама R попала в окно целиком, и после неё есть ещё пустое место в конце окна, то окно можно сдвинуть влево на эту длину.

Во-вторых, пусть сумма длин реклам в окне равна S. Если S > T+1, то нам можно сдвинуться влево ещё на S-(T+1).

Заметим ещё, что если в ответе получилось отрицательное время, то ответом должен быть ноль (в условии требуется неотрицательный ответ).

Пример решения на языке Python:

```
n = int(input())
T = int(input())
t1 = [0] * n
t2 = [0] * n
for i in range(n):
   t1[i], t2[i] = map(int, input().split())
R = 0
```

```
cur = 0
for L in range(n):
   R = max(R, L)
   while R+1 < n and t1[R+1] < t1[L] + 3600 and cur+t2[R]-t1[R] <= T:
      cur += t2[R] - t1[R]
      R += 1
   ads = cur + min(t2[R], t1[L] + 3600) - t1[R]
   oldt1R = t1[R]
   if ads > T:
      if t2[R] < t1[L] + 3600:
         t1[L] -= t1[L] + 3600 - t2[R]
      if ads > T + 1:
         shift = ads - (T + 1)
         t1[L] -= shift
      t1[L] = max(t1[L], 0)
      print(t1[L])
      break
   if cur > 0:
      cur -= t2[L] - t1[L]
else:
   print(-1)
```

Здесь, как и в прошлой программе, else относится к for, а не к if.

Задача 4. Зарядка робота

Первую подзадачу можно решить так. Перебираем все числа от n+1 до n+k, для каждого такого числа a вычисляем НОК(a, n) перебором чисел a, a+1 и т.д.

Во второй подзадаче также перебираем числа от n+1 до n+k, но НОК вычисляем по формуле a·n/НОД(a,n), где наибольший общий делитель вычисляем алгоритмом Евклида.

В третьей подзадаче заметим, что НОД(n, n+t) = НОД(n, t). Поэтому для a=n+t, где $1 \le t \le k$, справедливо:

НОК(n, a) = n·(n+t) / НОД(n, t) = n^2/НОД(n, t) + n·t/НОД(n, t)

Пусть g – максимальный делитель n в диапазоне от 1 до k. При этом НОД(n, g) = g. Тогда, если взять t = g, то формула превращается в $n^2/g + n$.

При любых других t значение НОД(n, t) < g. Действительно, допустим, что НОД(n, t) ≥ g. Больше k это значение быть не может, иначе бы было t > k. А раз оно не превышает k и является делителем n, то оно бы и было взято в качестве g. Также заметим, что n·t/НОД(n, t) ≥ n.

Тогда получается, что n^2/НОД(n, t) + n·t/НОД(n, t) > $n^2/g + n$. Таким образом, выбор g является оптимальным, а ответом будет n+g.

Остаётся найти максимальный делитель n, не превосходящий k. В третьей подзадаче можно просто перебирать все возможные делители n за O(n). В полном решении заметим, что получить все делители числа n можно циклом не до n, а до корня квадратного из n. Пример такого решения на языке Python:

```
n = int(input())
k = int(input())
g = 1
i = 1
while (i * i <= n and i <= k):
    if (n % i == 0):
        if(i > g):
            g = i;
        if(n // i > g and n // i < k):
            g = n // i
    i += 1
print(n + g)
```

Можно решить эту задачу на полный балл, даже если и не догадаться, что нужен именно максимальный делитель n от 1 до k. Пусть минимальное НОК равно $d_1 \cdot d_2 \cdot d_3$, где d_1 – какой-то делитель числа n, d_2 – наибольший общий делитель чисел n и n+p, d_3 – делитель числа n+p.

Циклом до корня переберём d_1. Значение d_2 равно n // d_1. Значение d_3 нам нужно как можно меньше, но оно не может быть меньше чем n+1. Тогда его можно найти как $(n+1)/d_2$ с округлением вверх, или $(n+d_2)//d_2$, где ‘//’ – деление нацело. Если при этом $d_2 \cdot d_3 \leq n+k$, то сравниваем $d_1 \cdot d_2 \cdot d_3$ с текущим значением минимального НОК, и если оно меньше, то заменяем на текущий вариант (также запоминая $d_2 \cdot d_3$). Пример такого решения на языке Python:

```
n = int(input())
k = int(input())
p = n + 1
min_lcm = n * (n + 1)
d1 = 1
while d1 * d1 <= n:
   if n % d1 == 0:
      d2 = n // d1
      d3 = (n + d2) // d2
      if d2 * d3 <= n + k and d1 * d2 * d3 <= min_lcm:
         p = d2 * d3
         min_lcm = d1 * d2 * d3
   d1 += 1
print(p)
```

При решении вторым способом на других языках есть одна проблема – значение $d_1 \cdot d_2 \cdot d_3$ может быть слишком большим даже для 64-битного типа данных. На C++ можно использовать нестандартный тип __int128, доступный в 64-битном компиляторе GNU C++. На других языках можно использовать длинную арифметику, в том числе из стандартных библиотек (класс BigInteger в Java и C#).

Задача 5. Порядок жизни

Первая подзадача решается тривиально, достаточно просто различить между собой два поля, одно из которых целиком заполнено нулями.

Во второй подзадаче можно написать наивное решение, где мы будем симулировать игру из каждого данного поля, формируя массив из целых чисел --- количество ходов, оставшихся до стабилизации поля. После вычисления этого массива нужно будет отсортировать полученные числа и получить искомую перестановку.

Для решения третьей задачи мы хотим однозначно сопоставить номера данных конфигураций самим конфигурациям. Это необходимо для того, чтобы в процессе симуляции мы могли понять, есть ли текущая конфигурация в нашем исходном наборе или нет. Для этого можно воспользоваться unordered_map в C++ и dict в Python.

Теперь заметим, что каждая конфигурация поля в ходе симуляции неизбежно приходит к стабильному состоянию. Запустим симулятор из первой данной конфигурации. Если в ходе симуляции мы встретим конфигурацию, которая уже есть в нашем наборе, то запомним для неё предыдущую. Затем повторим это для второй, третьей и так далее конфигураций поля. Ответ восстанавливается от конца к началу, начиная с конфигурации, для которой мы не нашли следующую (значит, она – последняя). Итоговая асимптотика составит $O(n \cdot m \cdot k)$.

Пример такого решения на языке Python (для получения полного балла нужно сдавать его на PyPy):

```
k = int(input())
n = int(input())
m = int(input())
```

```
def getNext(a):
   b = [[0] * m for _ in range(n)]
   for i in range(n):
      for j in range(m):
         t = 0
         for ii in (i-1, i, i+1):
            if ii < 0 or ii >= n:
               continue
            for jj in (j-1, j, j+1):
               if jj < 0 or jj >= m or ii == i and jj == j:
                  continue
               t += a[ii][jj]
         if a[i][j] == 1:
               b[i][j] = 1 if t == 2 or t == 3 else 0
         else:
            b[i][j] = 1 if t == 3 else 0
   return b

f = [None] * k
d = dict()
for i in range(k):
   f[i] = [[int(x) for x in input().split()] for _ in range(n)]
   s = str(f[i])
   d[s] = i
prev = [-1] * k
lastcell = -1
for i in range(k):
   p = f[i]
   while True:
      q = getNext(p)
      if q == p:
         lastcell = i
         break
      s = str(q)
      if s in d:
         prev[d[s]] = i
         break
      p = q
ans = [lastcell + 1]
i = lastcell
while prev[i] != -1:
   i = prev[i]
   ans.append(i + 1)
print(*ans[::-1])
```